Name: **Rahul Nalawade** Roll No: **201351017**

Indian Institute of Information Technology, Vadodara

CS308 – Introduction to Artificial Intelligence

Assignment #03

1. Implement Logistic Regression model with Gradient Descent method for minimizing E_{in}.

#Theory:

- **Logistic regression algorithm:**
  step 1 – initialize the weights at t=0 to w(0)
  step 2 – for t = {0, 1, 2, ..., B} B:= an upper bound for no of iterations
  compute the gradient

$$\nabla E_{in} = -\frac{1}{N} \sum_{n=1}^{N} \frac{y_n x_n}{1 + \exp(y_n * w^T(t) * x_n)}$$

  step 3 – update weights: $w(t+1) = w(t) - \eta \nabla E_{in}$
  step 4 – iterate until it is time to stop (t = B)
  step 5 – return final weights **w**

  Note: $\eta = 0.1$ (learning rate)
  randomization in initial weights helps (*Stochastic Gradient Descent*)

- **Minimizing E_{in}:**

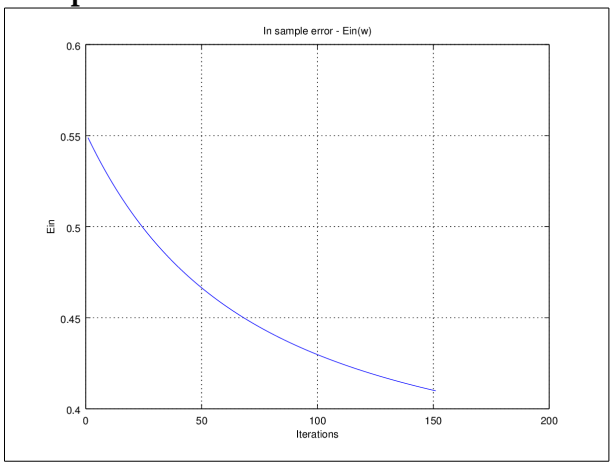
$$E_{in}(w) = \frac{1}{N} \sum_{n=1}^{N} \ln(1 + \exp(-y_n * w^T * x_n))$$

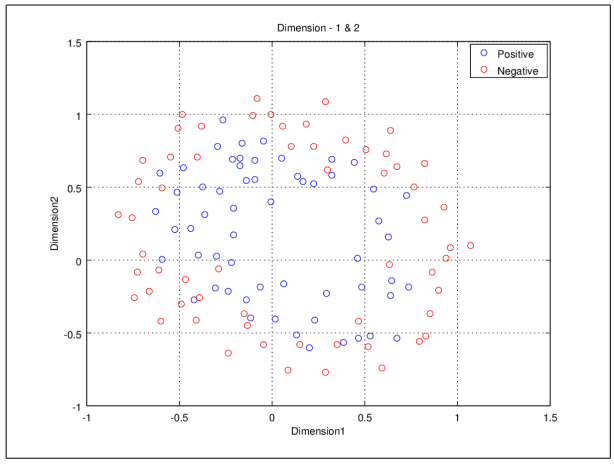
#Source Code:

```
→ assign31.m
clear all;
close all;

M = load('-ascii', 'ex2data2.txt');

[N A] = size(M);
A--;

iterations = 150;

weights = logistic_reg(M, iterations);

Ein = gradient_des(M,weights);

plot(Ein);
hold on
grid on
```

```
xlabel('Iterations');
ylabel('Ein');

title('In sample error - Ein(w)');
print('Ein - Gradient Descent.png','-dpng');
hold off

l = length(weights);

G = weights(l,1:A+1);

plotting_cp(M,G);
```

#Output:





→ logistic_reg.m

```
function W = logistic_reg(M,Q)

% logistic_reg(matrix,Q) trains on the data given in the form of matrix
% M = [attribute1 attribute2 . . . attributeN targetFunction];
% Q = No of training iterations for w(t) where t = {1, 2, 3, ..., Q}
% produces hypothesis weights matrix where W = [w(1); w(2); w(3); ...; w(Q)] and
% w(t) = {w0, w1, w2, . . ., wN} matrix for each t

[N, A] = size(M);
A--;

X = ones(N,1);
X = [X M(1:N,1:A)];
T = M(1:N,A+1);

t = 1;

g = 0.1;
```

```
w(t,1:A+1) = rand(1,A+1);

for t=1:Q

      for i=1:N
            E(i,1:A+1) = (T(i)*X(i,1:A+1)) / (1 +
exp(T(i)*w(t,1:A+1)*transpose(X(i,1:A+1))));
      end

      d_Ein(t,1:A+1) = (-1/N)*sum(E);

      w(t+1,1:A+1) = w(t,1:A+1) - g*d_Ein(t,1:A+1);
end
W = w;
```

→ gradient_des.m

```
function E_IN = gradient_des(M,weights)

% gradient_des(matrix,weight_matrix) computes the in sample error - Ein using
% M = [attribute1 attribute2 . . . attributeN targetFunction];
% w = Matrix containing improvised weight vectors
% produces Ein corresponding to each weight vector from w

[N A] = size(M);

A--;

X = ones(N,1);
X = [X M(1:N,1:A)];
T = M(1:N,A+1);

P = length(weights);

for t=1:P

      w = weights(t,1:A+1);

      for i=1:N

            E(i) = log(1 + exp(-1*T(i)*w*transpose(X(i,1:A+1))));

      end

      Ein(t) = (1/N)*sum(E);

end
```

```
E_IN = Ein;
```

→ plotting_cp.m

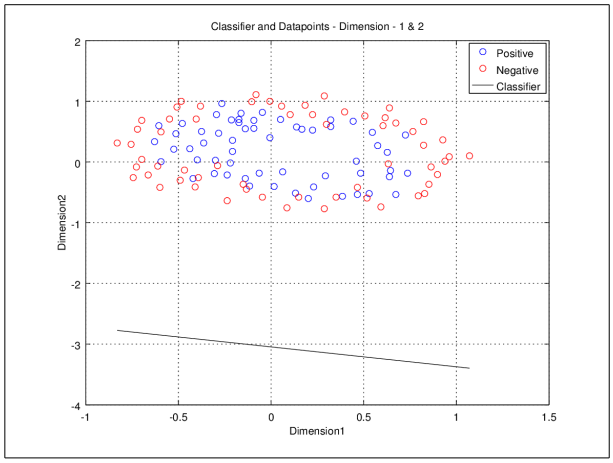
```
function plotting_cp(M,w)

% plots the data points with the Classifier Hyper plane base on the w
% M = [attribute1 attribute2 . . . attributeN targetFunction];
% w = [ w0 w1 w2 ... wd ]  d = VC dimension



% NOT A GENERIC ONE, WORK IS IN PROGRESS, BUT WORKS COMPLETELY FINE FOR A=2

[len, A] = size(M);

A--;

X = [M(1:len,1:A)];

T = [M(1:len,A+1)];

p=0;
n=0;

for i=1:len

      if(T(i) == 1)
            Positive(++p) = i;
      else
            Negative(++n) = i;
      end
end

if (A > 3)

      for j=1:A

            switch j

            case A-1,
                  d1 = j;
                  d2 = j+1;
                  d3 = 1;

            case A,
                  d1 = j;
                  d2 = 1;
```

```
                d3 = 2;

            otherwise,
                d1 = j;
                d2 = j+1;
                d3 = j+2;
            end

            s0 = strvcat(["Classifier and Datapoints - Dimension - ",int2str(d1),",
",int2str(d2)," & ",int2str(d3)]);

            s1 = strvcat(["Dimension",int2str(d1)]);
            s2 = strvcat(["Dimension",int2str(d2)]);
            s3 = strvcat(["Dimension",int2str(d3)]);

            for k=1:length(Positive)

                xp(k) = X((Positive(k)),d1);
                yp(k) = X((Positive(k)),d2);
                zp(k) = X((Positive(k)),d3);
            end

            for k=1:length(Negative)

                xn(k) = X((Negative(k)),d1);
                yn(k) = X((Negative(k)),d2);
                zn(k) = X((Negative(k)),d3);
            end

            xc = linspace(min(X(1:len,d1)), max(X(1:len,d1)),1000);
            yc = linspace(min(X(1:len,d2)), max(X(1:len,d2)),1000);

            [XC, YC] = meshgrid(xc,yc);

            ZC = (w(1)/w(d3+1))*1 - (w(d1+1)/w(d3+1)).*XC - (w(d2+1)/w(d3+1)).*YC;

            plot3(xp,yp,zp,'bo',xn,yn,zn,'ro')
            hold on
            mesh(XC,YC,ZC);
            grid on
            xlabel(s1);
            ylabel(s2);
            zlabel(s3);
            legend('Positive','Negative')
            title(s0);
            print(strvcat([ s0,".png"]),'-dpng');
```

```
      end

elseif(A == 3)

      d1 = 1;
      d2 = 2;
      d3 = 3;

      s0 = strvcat(["Classifier and Datapoints - Dimension - ",int2str(d1),",
",int2str(d2)," & ",int2str(d3)]);

      s1 = strvcat(["Dimension",int2str(d1)]);
      s2 = strvcat(["Dimension",int2str(d2)]);
      s3 = strvcat(["Dimension",int2str(d3)]);

      for k=1:length(Positive)

            xp(k) = X((Positive(k)),d1);
            yp(k) = X((Positive(k)),d2);
            zp(k) = X((Positive(k)),d3);
      end

      for k=1:length(Negative)

            xn(k) = X((Negative(k)),d1);
            yn(k) = X((Negative(k)),d2);
            zn(k) = X((Negative(k)),d3);
      end

      xc = linspace(min(X(1:len,d1)), max(X(1:len,d1)),1000);
      yc = linspace(min(X(1:len,d2)), max(X(1:len,d2)),1000);

      [XC, YC] = meshgrid(xc,yc);

      ZC = (w(1)/w(4))*1 - (w(2)/w(4)).*XC - (w(3)/w(4)).*YC;

      figure2
      plot3(xp,yp,zp,'bo',xn,yn,zn,'ro')
      hold on
      mesh(XC,YC,ZC);
      grid on
      xlabel(s1);
      ylabel(s2);
      zlabel(s3);
      legend('Positive','Negative')
      title(s0);
      print(strvcat([ s0,".png"]),'-dpng');
```

```
elseif(A == 2)

      d1 = 1;
      d2 = 2;

      s0 = strvcat(["Classifier and Datapoints - Dimension - ",int2str(d1)," &
",int2str(d2)]);

      s1 = strvcat(["Dimension",int2str(d1)]);
      s2 = strvcat(["Dimension",int2str(d2)]);

      for k=1:length(Positive)

            xp(k) = X((Positive(k)),d1);
            yp(k) = X((Positive(k)),d2);
      end

      for k=1:length(Negative)

            xn(k) = X((Negative(k)),d1);
            yn(k) = X((Negative(k)),d2);

      end

      XC = linspace( min(X(1:len,d1)), max(X(1:len,d1)),1000);

      YC = (-w(1)/w(3))*1 - (w(2)/w(3)).*XC;


      plot(xp,yp,'bo',xn,yn,'ro',XC,YC,'k')
      hold on
      grid on
      xlabel(s1);
      ylabel(s2);
      legend('Positive','Negative','Classifier')
      title(s0);
      print(strvcat([ s0,".png"]),'-dpng');



else

      d1 = 1;


      s0 = strvcat(["Classifier and Datapoints - Dimension - ",int2str(d1)]);
```

```
        s1 = strvcat(["Dimension",int2str(d1)]);


        for k=1:length(Positive)

              xp(k) = X((Positive(k)),d1);

        end

        for k=1:length(Negative)

              xn(k) = X((Negative(k)),d1);

        end

        plot(xp,'bo',xn,'ro')
        hold on
        grid on
        xlabel(s1);
        legend('Positive','Negative')
        title(s0);
        print(strvcat([ s0,".png"]),'-dpng');

end
```



# # #